Pipelined Radix-2^k Feedforward FFT Architectures

Mario Garrido, *Member, IEEE*, J. Grajal, M. A. Sánchez, and Oscar Gustafsson, *Senior Member, IEEE*

Abstract—**The appearance of radix-2^2 was a milestone in the design of pipelined FFT hardware architectures. Later, radix-2^2 was extended to radix-2^k. However, radix-2^k was only proposed for single-path delay feedback (SDF) architectures, but not for feedforward ones, also called multi-path delay commutator (MDC). This paper presents the radix-2^k feedforward (MDC) FFT architectures. In feedforward architectures radix-2^k can be used for any number of parallel samples which is a power of two. Furthermore, both decimation in frequency (DIF) and decimation in time (DIT) decompositions can be used. In addition to this, the designs can achieve very high throughputs, which makes them suitable for the most demanding applications. Indeed, the proposed radix-2^k feedforward architectures require fewer hardware resources than parallel feedback ones, also called multi-path delay feedback (MDF), when several samples in parallel must be processed. As a result, the proposed radix-2^k feedforward architectures not only offer an attractive solution for current applications, but also open up a new research line on feedforward structures.**

Index Terms—**Fast Fourier transform (FFT), multipath delay commutator (MDC), pipelined architecture, radix-2^k, VLSI.**

I. Introduction

THE FAST Fourier transform (FFT) is one of the most important algorithms in the field of digital signal processing. It is used to calculate the discrete Fourier transform (DFT) efficiently. In order to meet the high performance and real-time requirements of modern applications, hardware designers have always tried to implement efficient architectures for the computation of the FFT. In this context, pipelined hardware architectures [1]–[24] are widely used, because they provide high throughputs and low latencies suitable for real time, as well as a reasonably low area and power consumption.

There are two main types of pipelined architectures: feedback (FB) and feedforward (FF). On the one hand, feedback architectures [1]–[14] are characterized by their feedback loops, i.e., some outputs of the butterflies are fed back to the memories at the same stage. Feedback architectures can be divided into single-path delay feedback (SDF) [1]–[6], which process a continuous flow of one sample per clock cycle, and multi-path delay feedback (MDF) or parallel feedback [7]–[14], which process several samples in parallel. On the other hand, feedforward architectures [4], [5], [15]–[19], also known as multi-path delay commutator (MDC) [4], do not have feedback loops and each stage passes the processed data to the next stage. These architectures can also process several samples in parallel.

In current real-time applications, the FFT has to be calculated at very high throughput rates, even in the range of Gigasamples per second. These high-performance requirements appear in applications such as orthogonal frequency division multiplexing (OFDM) [9]–[12], [22] and ultra wideband (UWB) [10]–[13]. In this context two main challenges can be distinguished. The first one is to calculate the FFT of multiple independent data sequences [22], [23]. In this case, all the FFT processors can share the rotation memory in order to reduce the hardware [22]. Designs that manage a variable number of sequences can also be obtained [23]. The second challenge is to calculate the FFT when several samples of the same sequence are received in parallel. This must be done when the required throughput is higher than the clock frequency of the device. In this case it is necessary to resort to FFT architectures that can manage several samples in parallel.

As a result, parallel feedback architectures, which had not been considered for several decades, have become very popular in the last few years [8]–[14]. Conversely, not very much attention has been paid to feedforward (MDC) architectures. This paradoxical fact, however, has a simple explanation. Originally, SDF and MDC architectures were proposed for radix-2 [2], [17] and radix-4 [3], [17]. Some years later, radix-2^2 was presented for the SDF FFT [4] as an improvement on radix-2 and radix-4. Next, radix-2^3 and radix-2^4, which enable certain complex multipliers to be simplified, were also presented for the SDF FFT. An explanation of radix-2^k SDF architectures can be found in [6]. Finally, the current need for high throughput has been meet by the MDF, which includes multiple interconnected SDF paths in parallel. However, radix-2^k had not been considered for feedforward architectures until the first radix-2^2 feedforward FFT architectures were proposed a few years ago [24].

In this work we present the radix-2^k feedforward FFT architectures. The proposed designs include radix-2^2, radix-2^3 and radix-2^4 architectures. The paper shows that radix-2^k can be used for any number of parallel samples which is a power of two. Accordingly, radix-2^k FFT architectures for 2, 4, and 8 parallel samples are presented. These architectures are shown to be more hardware-efficient than previous feedforward and parallel feedback designs in the literature. This makes them very attractive for the computation of the FFT in the most demanding applications.

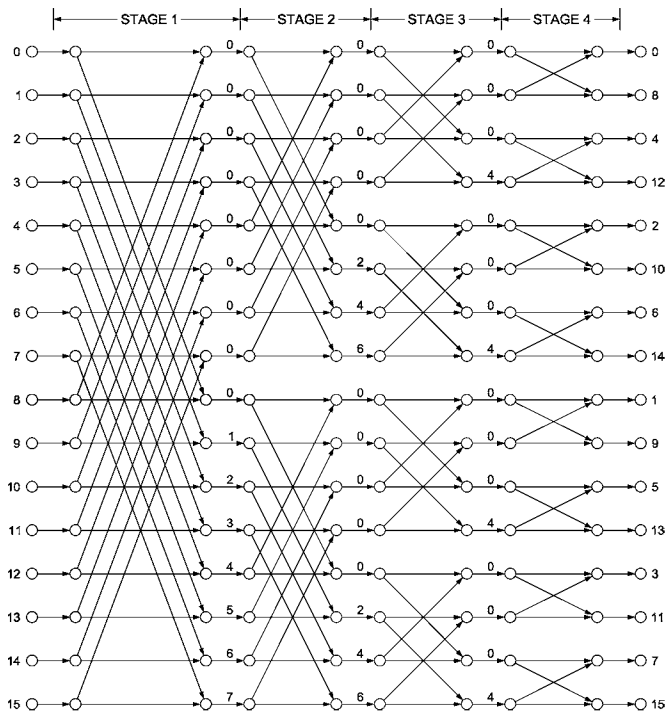

Fig. 1. Flow graph of the 16-point radix-2 DIF FFT.

This paper is organized as follows. Section II explains the radix-2^2 FFT algorithm and Section III shows how to design radix-2^2 FFT architectures. As a result, the pipelined radix-2^2 feedforward FFT architectures are presented in Section IV, where architectures for different number of parallel samples using DIF and DIT decompositions are proposed. In Section V, the results are extended to radix-2^k and feedforward FFT architectures for radix-2^3 and radix-2^4 are presented. In Section VI, the proposed designs are compared to previous ones and in Section VII experimental results are provided. Finally, the main contributions of this work are summarized in Section VIII.

II. Radix-2^2 FFT Algorithm

The N-point DFT of an input sequence $x[n]$ is defined as

$$X[k] = \sum_{n=0}^{N-1} x[n] W_N^{nk}, \quad k = 0, 1, \ldots, N-1 \qquad (1)$$

where $W_N^{nk} = e^{-j(2\pi/N)nk}$.

When N is a power of two, the FFT based on the Cooley-Tukey algorithm [25] is most commonly used in order to compute the DFT efficiently. The Cooley-Tukey algorithm reduces the number of operations from $O(N^2)$ for the DFT to $O(N \log_2 N)$ for the FFT. In accordance with this, the FFT is calculated in a series of $n = \log_\rho N$ stages, where ρ is the base of the radix, r, of the FFT, i.e., $r = \rho^\alpha$.

Figs. 1 and 2 show the flow graphs of 16-point radix-2 and radix-2^2 FFTs, respectively, decomposed using decimation in frequency (DIF) [26]. At each stage of the graphs, $s \in \{1, \ldots, n\}$, butterflies and rotations have to be calculated.

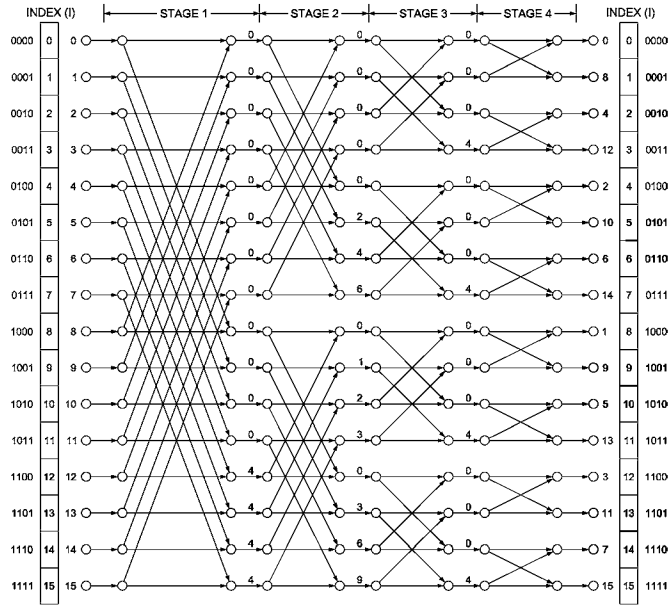

Fig. 2. Flow graph of the 16-point radix-2^2 DIF FFT.

The lower edges of the butterflies are always multiplied by -1. These -1 are not depicted in order to simplify the graphs.

The numbers at the input represent the index of the input sequence, whereas those at the output are the frequencies, k, of the output signal $X[k]$. Finally, each number, ϕ, in between the stages indicates a rotation by

$$W_N^\phi = e^{-j\frac{2\pi}{N}\phi}. \qquad (2)$$

As a consequence, samples for which $\phi = 0$ do not need to be rotated. Likewise, if $\phi \in [0, N/4, N/2, 3N/4]$ the samples must be rotated by 0°, 270°, 180°, and 90°, which correspond to complex multiplications by 1, $-j$, -1 and j, respectively. These rotations are considered trivial, because they can be performed by interchanging the real and imaginary components and/or changing the sign of the data.

Radix-2^2 is based on radix-2 and the flow graph of a radix-2^2 DIF FFT can be obtained from the graph of a radix-2 DIF one. This can be done by breaking down each angle, ϕ, at odd stages into a trivial rotation and a non-trivial one, ϕ', where $\phi' = \phi \bmod N/4$, and moving the latter to the following stage. This is possible thanks to the fact that in the radix-2 DIF FFT the rotation angles at the two inputs of every butterfly, ϕ_A and ϕ_B, only differ by 0 or $N/4$. Thus, if $\phi_A = \phi'$ and $\phi_B = \phi' + N/4$, the rotation ϕ' is moved to the following stage in accordance with

$$Ae^{-j\frac{2\pi}{N}\phi'} \pm Be^{-j\frac{2\pi}{N}(\phi'+N/4)} = [A \pm (-j)B] \cdot e^{-j\frac{2\pi}{N}\phi'} \qquad (3)$$

where the first side of (3) represents the computations using radix-2 and the second one using radix-2^2, A and B being the input data of the butterfly. In radix-2, A and B are rotated before the butterfly is computed, whereas in radix-2^2 B is rotated by the trivial rotation $-j$ before the butterfly, and the remaining rotation is carried out after the butterfly. Consequently, rotations by ϕ' can be combined with those rotations of the following

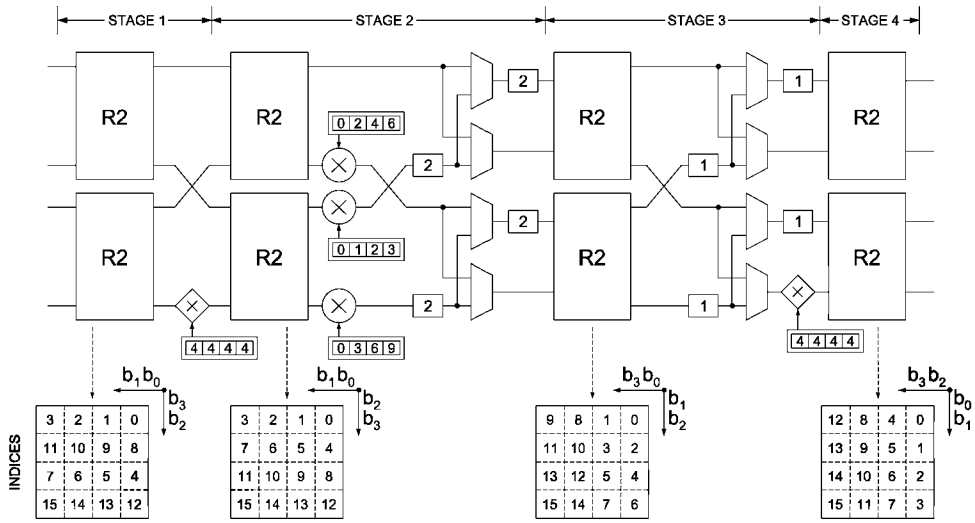

Fig. 3. Proposed 4-parallel radix-2^2 feedforward architecture for the computation of the 16-point DIF FFT.

TABLE I
PROPERTIES OF THE RADIX-2^2 FFT ALGORITHM FOR DIF AND DIT

Properties Radix-2^2	DIF	DIT
Butterflies	b_{n-s}	b_{n-s}
Trivial rotations (odd s)	$b_{n-s} \cdot b_{n-s-1} = 1$	$b_{n-s} \cdot b_{n-s-1} = 1$
Non-trivial rotations (even s)	$b_{n-s+1} + b_{n-s} = 1$	$b_{n-s-1} + b_{n-s-2} = 1$

stage. This derivation of radix-2^2 from radix-2 can be observed in Figs. 1 and 2 for the particular case of $N = 16$.

Analogously, the radix-2^2 DIT FFT can be derived from the radix-2 DIT FFT. Contrary to DIF, for DIT the non-trivial rotations ϕ' are moved to the previous stage instead of the following one.

III. DESIGNING RADIX-2^2 FFT ARCHITECTURES

The proposed architectures have been derived using the framework presented in [24]. The design is based on analyzing the flow graph of the FFT and extracting the properties of the algorithm. These properties are requirements that any hardware architecture that calculates the algorithm must fulfill. The properties of the radix-2^2 FFT are shown in Table I. The following paragraphs explain these properties and how they are obtained.

The properties depend on the index of the data, $I \equiv b_{n-1}, \ldots, b_1, b_0$, where $(\equiv)$ will be used throughout the paper to relate both the decimal and the binary representations of a number. This index is included in Fig. 2 both in decimal and in binary.

On the one hand, the properties related to the butterfly indicate which samples must be operated together in the butterflies. This condition is b_{n-s} both for DIF and DIT decompositions and means that at any stage of the FFT, s, butterflies operate in pairs of data whose indices differ only in bit b_{n-s}, where $n = \log_2 N$ is the number of stages of the FFT. In Fig. 2 it can be observed that at the third stage, $s = 3$, data with indices $I = 12 \equiv 1100$ and $I' = 14 \equiv 1110$ are processed together by a butterfly. These indices differ in bit b_1, which meets b_{n-s}, since $n = \log_2 N = \log_2 16 = 4$ and, thus, $b_{n-s} = b_{4-3} = b_1$.

On the other hand, there are two properties for rotations. At odd stages of the radix-2^2 DIF FFT only those samples whose index fulfills $b_{n-s} \cdot b_{n-s-1} = 1$ have to be rotated. These rotations are trivial and the symbol $(\cdot)$ indicates the logic AND function. For the 16-point radix-2^2 FFT in Fig. 2 only samples with indices 12, 13, 14, and 15 must be rotated at the first stage. For these indices $b_3 \cdot b_2 = 1$ is fulfilled, meeting the property $b_{n-s} \cdot b_{n-s-1} = 1$, since $n = 4$ and $s = 1$. Conversely, at even stages rotations are non-trivial and they are calculated over indexed data for which $b_{n-s+1} + b_{n-s} = 1$, where the symbol $(+)$ indicates the logic OR function.

IV. RADIX-2^2 FEEDFORWARD FFT ARCHITECTURES

This section presents the radix-2^2 feedforward architectures. First, a 16-point 4-parallel radix-2^2 feedforward FFT architecture is explained in depth in order to clarify the approach and show how to analyze the architectures. Then, radix-2^2 feedforward architectures for different number of parallel samples are presented.

Fig. 3 shows a 16-point 4-parallel radix-2^2 feedforward FFT architecture. The architecture is made up of radix-2 butterflies (R2), non-trivial rotators $(\otimes)$, trivial rotators, which are diamond-shaped, and shuffling structures, which consist of buffers and multiplexers. The lengths of the buffers are indicated by a number.

The architecture processes four samples in parallel in a continuous flow. The order of the data at the different stages is shown at the bottom of the figure by their indices, together with the bits b_i that correspond to these indices. In the horizontal, indexed samples arrive at the same terminal at different time instants, whereas samples in the vertical arrive at the same time at different terminals. Finally, samples flow from left to right. Thus, indexed samples (0, 8, 4, 12) arrive in parallel at the inputs of the circuit at the first clock cycle, whereas indexed samples (12, 13, 14, 15) arrive at consecutive clock cycles at the lower input terminal.

Taking the previous considerations into account, the architecture can be analyzed as follows. First, it can be observed

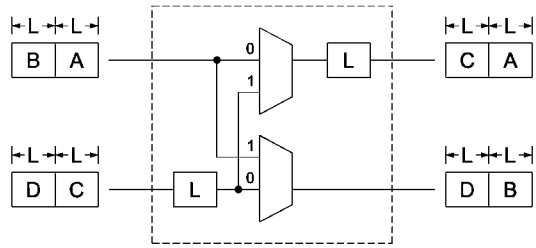

Fig. 4. Circuit for data shuffling.

that butterflies always operate in pairs of samples whose indices differ in bit b_{n-s}, meeting the property in Table I. For instance, the pairs of data that arrive at the upper butterfly of the first stage are: (0, 8), (1, 9), (2, 10), and (3, 11). The binary representation of these pairs of numbers only differ in b_3. As, $n = 4$ and $s = 1$ at the first stage, $b_{n-s} = b_{4-1} = b_3$, so the condition is fulfilled. This property can also be checked for the rest of the butterflies in a similar way.

Second, Table I shows that rotations at odd stages are trivial and only affect samples whose indices fulfill $b_{n-s} \cdot b_{n-s-1} = 1$. By particularizing this condition for the first stage, $b_3 \cdot b_2 = 1$ is obtained. In the architecture shown in Fig. 3 the indices that fulfill this condition are those of the lower edge and, thus, a trivial rotator is included at that edge. On the other hand, the condition for non-trivial rotations at even stages is $b_{n-s+1} + b_{n-s} = 1$, $b_3 + b_2 = 1$ being for the second stage. As $b_3 + b_2 = 0$ for all indexed samples at the upper edge of the second stage, this edge does not need any rotator. Conversely, for the rest of edges $b_3 + b_2 = 1$, so they include non-trivial rotators.

The rotation memories of the circuit store the coefficients ϕ of the flow graph. It can be seen that the coefficient associated to each index is the same as that in the flow graph of Fig. 2. For instance, at the flow graph the sample with index $I = 14$ has to be rotated by $\phi = 6$ at the second stage. In the architecture shown in Fig. 3 the sample with index $I = 14$ is the third one that arrives at the lower edge of the second stage. Thus, the third position of the rotation memory of the lower rotator stores the coefficient for the angle $\phi = 6$.

Thirdly, the buffers and multiplexers carry out data shuffling. These circuits have already been used in previous pipelined FFT architectures [4], [17]–[20], and Fig. 4 shows how they work. For the first L clock cycles the multiplexers are set to "0", L being the length of the buffers. Thus, the first L samples from the upper path (set A) are stored in the output buffer and the first L samples from the lower path (set C) are stored in the input buffer. Next, the multiplexer changes to "1", so set C passes to the output buffer and set D is stored in the input buffer. At the same time, sets A and B are provided in parallel at the output. When the multiplexer commutes again to "0", sets C and D are provided in parallel. As a result, sets B and C are interchanged.

Finally, the control of the circuit is very simple: As the multiplexers commute every L clock cycles and L is a power of two, the control signals of the multiplexers are directly obtained from the bits of a counter.

Fig. 5 shows the proposed radix-2^2 feedforward architectures for the computation of the 64-point DIF FFT. Figs. 5(a)–(c) show the cases of 2-parallel, 4-parallel, and 8-parallel samples, respectively. These circuits can be analyzed as has been done for the architecture in Fig. 3. For this purpose, the order of the samples at every stage has been added at the bottom of the architectures.

As can be seen in Fig. 5, in the proposed architectures the number of butterflies depends on to the number of samples in parallel, $P = 2^p$. For any P-parallel N-point FFT the number of butterflies is $P/2 \cdot \log_2 N = P \cdot \log_4 N$. Therefore, the number of complex adders is $2P \cdot \log_4 N$. Likewise, the number of rotators is $3P/4 \cdot (\log_4 N - 1)$. The only exception is for $P = 2$. In this case, the number of rotators is $2 \cdot (\log_4 N - 1)$.

The proposed architectures can process a continuous flow of data. The throughput in samples per clock cycle is equal to the number of samples in parallel $P = 2^p$, whereas the latency is proportional to the size of the FFT divided by the number of parallel samples, i.e., N/P. Thus, the most suitable architecture for a given application can be selected by considering the throughput and latency that the application demands. Indeed, the number of parallel samples can be increased arbitrarily, which assures that the most demanding requirements are met.

Finally, the memory size does not increase with the number of parallel samples. For the architectures shown in Fig. 5, the shuffling structure at any stage $s \in [p, n-1]$ requires $P = 2^p$ buffers of length $L = N/2^{s+1}$. According to this, the total sample memory of the architectures is

$$\sum_{s=p}^{n-1} 2^p \cdot L = \sum_{s=p}^{\log_2 N - 1} 2^p \cdot \frac{N}{2^{s+1}} = N - 2^p = N - P. \quad (4)$$

Therefore, a total sample memory of N addresses is enough for the computation of an N-point FFT independently of the degree of parallelism of the FFT. Indeed, the total memory of $N - P$ addresses that the proposed architectures require is the minimum amount of memory for an N-point P-parallel FFT.

Sometimes input samples are provided to the FFT in natural order and output frequencies are also required in natural order [27], [28]. Under these circumstances, reordering circuits are required before and after the FFT to adapt the input and output orders [27], [28]. For the proposed radix-2^2 feedforward FFTs the memory requirements for natural I/O depend on the FFT size and on the number of parallel samples. For a P-parallel N-point FFT a total memory of size $N - N/P$ is enough to carry out the input reordering, whereas a total memory of size N is enough for the output reordering [24].

The proposed approach can also be used to derive radix-2^2 feedforward architectures FFT for DIT. In this case, the properties for DIT in Table I must be considered. Accordingly, Fig. 6 shows a 4-parallel radix-2^2 feedforward architecture for the computation of the 64-point DIT FFT. This architecture can be compared with the DIF version in Fig. 5(b). It can be noted that both DIF and DIT architectures use the same number of hardware components. Nevertheless, the layout of the components is different. For any number of parallel samples, DIF and DIT architectures also require the same number of components.

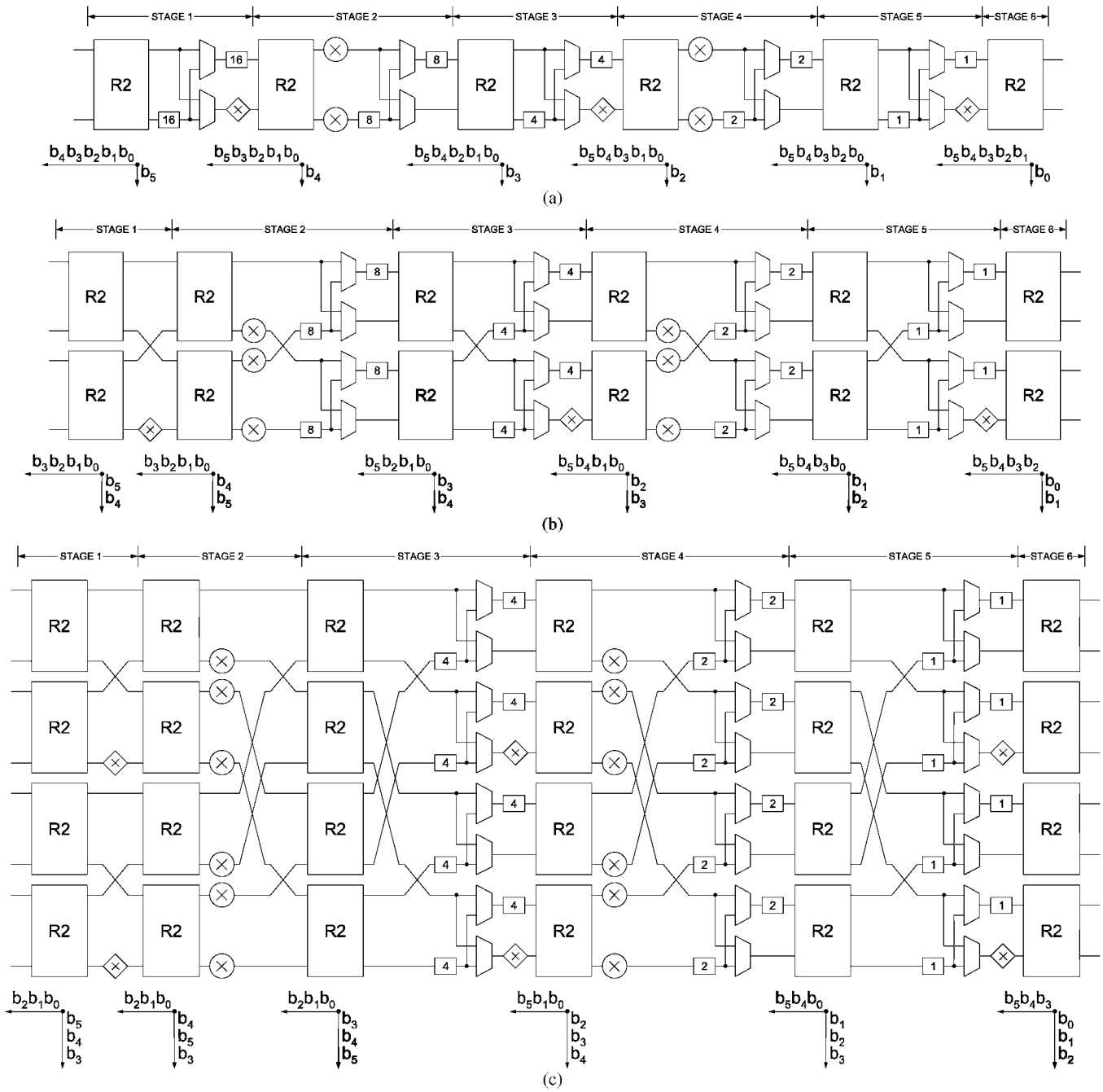

Fig. 5. Proposed radix-2^2 feedforward architectures for the computation of the 64-point DIF FFT. (a) 2-parallel radix-2^2 feedforward FFT. (b) 4-parallel radix-2^2 feedforward FFT. (c) 8-parallel radix-2^2 feedforward FFT.

V. Extension to Radix-2^k

Table II shows the properties for the radix-2^3 and radix-2^4 FFT algorithms. As for radix-2^2, these properties have been obtained directly from the flow graphs of the algorithms. The conditions for butterflies are the same for all stages of the FFT, whereas the conditions for rotations depend on the stage, s. Rotations are classified into trivial (T), non-trivial (NT), and rotations by W_8 or W_{16}. Rotations by W_8 and W_{16} are not-trivial, but include a reduced set of angles [29]. According to (2), rotations by W_8 only consider angles that are multiples of $\pi/4$, whereas W_{16} only includes multiples of $\pi/8$. This allows for the simplification of the rotators that carry out the rotations. For this purpose, different techniques have been proposed in the literature. They include the use of trigonometric identities [30], the representation of the coefficients in canonical signed digit (CSD) [9] and the scaling of the coefficients [29]. Finally, in the table $i \in \mathbb{Z}$ and, thus, for radix-2^k the type of rotation repeats every k stages.

Fig. 7(a) and (b) show the proposed radix-2^3 feedforward architectures, respectively for 2 and 4 samples in parallel. It can be observed that radix-2^3 feedforward architectures only require general non-trivial rotators every three stages. Additionally, the

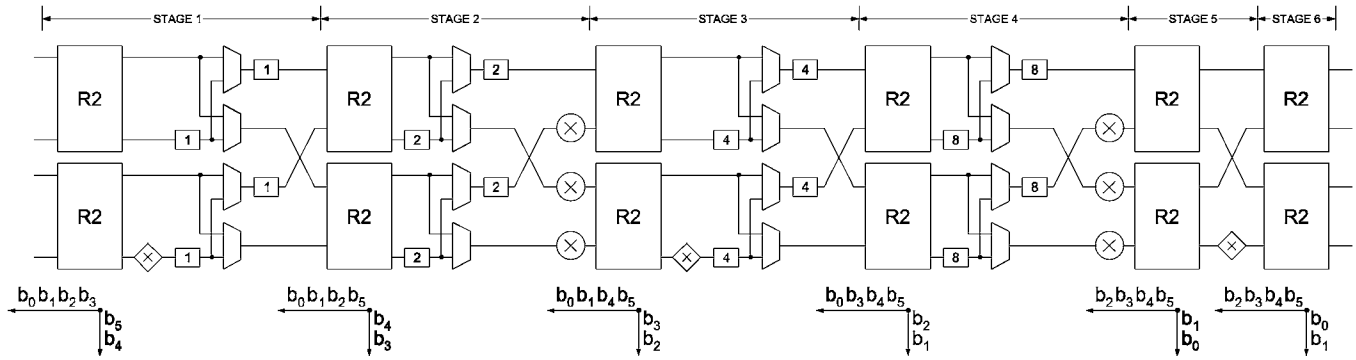

Fig. 6. Proposed 4-parallel radix-2^2 feedforward architecture for the computation of the 64-point DIT FFT.

TABLE II
PROPERTIES OF THE RADIX-2^3 AND RADIX-2^4 FFT ALGORITHMS FOR DIF AND DIT

Properties Radix-2^3		DIF		DIT	
Butterflies	$\forall s$	b_{n-s}		b_{n-s}	
Rotations	$s=3i+1$	$b_{n-s}\cdot(b_{n-s-1}+b_{n-s-2})=1$	W_8	$b_{n-s}\cdot b_{n-s-1}=1$	T
	$s=3i+2$	$b_{n-s}\cdot b_{n-s-1}=1$	T	$b_{n-s-1}\cdot(b_{n-s}+b_{n-s+1})=1$	W_8
	$s=3i+3$	$b_{n-s+2}+b_{n-s+1}+b_{n-s}=1$	NT	$b_{n-s-1}+b_{n-s-2}+b_{n-s-3}=1$	NT
Properties Radix-2^4		DIF		DIT	
Butterflies	$\forall s$	b_{n-s}		b_{n-s}	
Rotations	$s=4i+1$	$b_{n-s}\cdot b_{n-s-1}=1$	T	$b_{n-s}\cdot b_{n-s-1}=1$	T
	$s=4i+2$	$(b_{n-s+1}+b_{n-s})\cdot(b_{n-s-1}+b_{n-s-2})=1$	W_{16}	$(b_{n-s+1}+b_{n-s})\cdot(b_{n-s-1}+b_{n-s-2})=1$	W_{16}
	$s=4i+3$	$b_{n-s}\cdot b_{n-s-1}=1$	T	$b_{n-s}\cdot b_{n-s-1}=1$	T
	$s=4i+4$	$b_{n-s+3}+b_{n-s+2}+b_{n-s+1}+b_{n-s}=1$	NT	$b_{n-s-1}+b_{n-s-2}+b_{n-s-3}+b_{n-s-4}=1$	NT

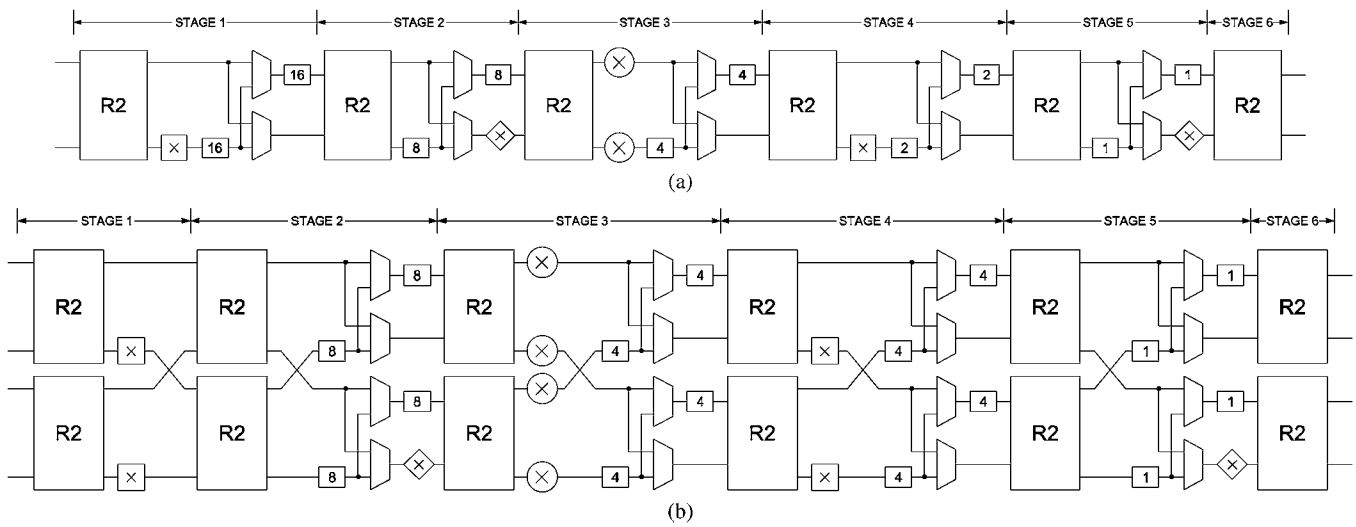

Fig. 7. Proposed radix-2^3 feedforward architectures for the computation of the 64-point DIF FFT. (a) 2-parallel radix-2^3 feedforward FFT. (b) 4-parallel radix-2^3 feedforward FFT.

architectures must calculate rotations by W_8, which are represented by squared-shaped rotators. Compared to the 2-parallel radix-2^2 feedforward architecture in Fig. 5(a), the 2-parallel radix-2^3 feedforward FFT in Fig. 7(a) has the same number of butterflies, rotators and total memory. However, some of the rotators for radix-2^3 calculate rotations by W_8, which can be simplified. Likewise, the 4-parallel radix-2^3 feedforward FFT in Fig. 7(b) includes fewer general rotators than the radix-2^2 one in Fig. 5(b).

The proposed radix-2^4 feedforward FFT architectures for $N = 256$ are shown in Fig. 8. The architectures also include square-shaped rotators, which carry out the rotations by W_{16}. Note that the 2-parallel radix-2^4 feedforward FFT is very similar to the 2-parallel radix-2^2 one, with the difference that general rotators every four stages in radix-2^2 are substituted by W_{16} rotators in radix-2^4. For 4-parallel samples, radix-2^4 also needs fewer general rotators than radix-2^2 and radix-2^3.

Architectures for a higher number of samples in parallel can also be obtained using radix-2^k. For a general case of a P-par-

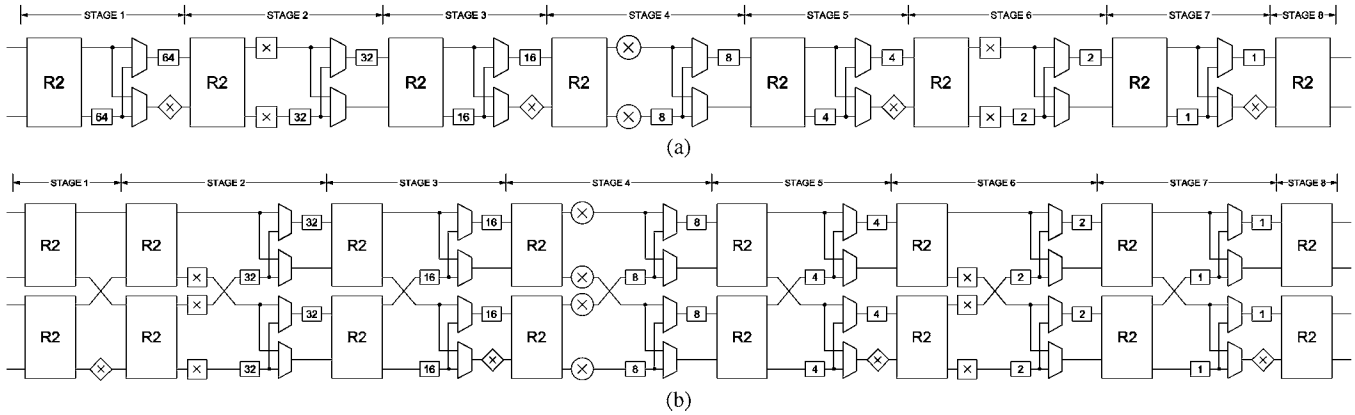

Fig. 8. Proposed radix-2^4 feedforward architectures for the computation of the 256-point DIF FFT. (a) 2-parallel radix-2^4 feedforward FFT. (b) 4-parallel radix-2^4 feedforward FFT.

allel radix-2^k N-point feedforward FFT, the number of complex adders is equal to

$$P \cdot \log_2 N \tag{5}$$

the number of general rotators can be calculated as

$$P \cdot \left(\frac{\log_2 N}{k} - 1 \right), \quad \text{if} \quad P < 2^k \tag{6}$$

$$\frac{2^k - 1}{2^k} \cdot P \cdot \left(\frac{\log_2 N}{k} - 1 \right), \quad \text{if} \quad P \geq 2^k \tag{7}$$

and the total memory is $N - P$. Likewise, the throughput is always equal to the number of parallel samples, P, and the latency is N/P.

Note that apart from general rotators, the architectures must include rotators that calculate the simpler non-trivial rotations by W_L, where $L = 2^k$ is the number of angles of the kernel. These kernels are W_8 and W_{16}, respectively for radix-2^3 and radix-2^4, which allow for efficient hardware implementations. Nevertheless, if k is larger, radix-2^k architectures include W_L kernels with larger number of angles. As a result, the implementation of these rotators becomes more complicated, being necessary to resort to general rotators in most cases.

Note also that the proposed radix-2^k feedforward FFT architectures can be used for any number of parallel samples, $P = 2^p$. Conversely, conventional feedforward architectures based on radix-r are only for $r \geq P$.

VI. Comparison and Analysis

Table III compares the proposed structures to other efficient pipelined architectures for the computation of an N-point FFT. The architectures are classified into 2-parallel, 4-parallel, and 8-parallel ones. The first two columns indicate the type of architecture and the radix. The rest of the table shows the trade-off between area and performance. On the one hand, area is measured in terms of the number of rotators, adders and memory. As different applications demand different input and output orders, circuits for data reordering before and after the FFTs are not considered in the comparison. Rotators are required for non-trivial rotations. In Table III they are classified into rotators for W_8 and W_{16}, and general rotators for other non-trivial rotations. The total number of rotators is also included.

On the other hand, performance is represented by throughput and latency. The latency is defined as the number of clock cycles that the architecture needs to process an input sequence, considering that it receives a continuous flow of data. Meanwhile, the throughput indicates the number of samples per clock cycle that are processed. In all architectures this throughput is equal to the number of samples that are processed in parallel.

Among 2-parallel architectures, the proposed radix-2^k feedforward FFTs require the same number of rotators, adders and memory as the radix-2 feedforward FFT [4]. However, some of the rotators in radix-2^3 and radix-2^4 FFTs can be simplified, as they only have to calculate rotations by W_8 and W_{16}. Compared to previous radix-2^4 parallel feedback architectures [11], the proposed radix-2^4 designs save 50% of the adders and reduce the memory requirements, while having the same number of rotators.

As regards 4-parallel architectures, the proposed radix-2^2 feedforward FFT and the radix-4 feedforward FFT [18], [19] require the lowest number of rotators, adders and memory among all the designs in the literature. Although radix-2^2 and radix-4 architectures require the same total number of hardware resources for 4-parallel samples, the layout of these resources is different: Whereas radix-2^2 admits circuits for data management and rotators between radix-2 butterflies, in radix-4 pairs of consecutive sets of radix-2 butterflies must necessarily be together in order to form the radix-4 butterfly.

By comparing the proposed 4-parallel radix-2^3 and radix-2^4 architectures to the 4-parallel radix-4 feedforward FFT [18], [19], it can be observed that radix-2^3 and radix-2^4 have the same number of adders and memory, but need fewer general rotators.

The proposed 4-parallel architectures also improve on parallel feedback architectures [9], [13]. The reason lies in the fact that in feedback FFTs the utilization ratio of butterflies is 50% and the parallelization cannot improve this ratio. Conversely, the proposed designs have a utilization ratio of 100%, so the number of adders is halved. Likewise, the number of rotators is reduced in the proposed architectures with respect to parallel feedback ones. Specifically the proposed radix-2^2 architecture

TABLE III
COMPARISON OF THE PROPOSED RADIX-2^k FEEDFORWARD ARCHITECTURES TO OTHER APPROACHES FOR THE COMPUTATION OF AN N-POINT FFT

PIPELINED ARCHITECTURE		AREA					PERFORMANCE	
		Rotators			Complex	Complex Data	Latency	Throughput
Type	Radix	Total	General	W_8 or W_{16}	Adders	Memory	(cycles)	(samples/cycle)
2-PARALLEL ARCHITECTURES								
FF (MDC)	Radix-2 [4]	$2(\log_4 N-1)$	$2(\log_4 N-1)$	0	$4(\log_4 N)$	N	$N/2$	2
FB (MDF)	Radix-2^2 [12]	$2(\log_4 N-1)$	$2(\log_4 N-1)$	0	$8(\log_4 N)$	N	$N/2$	2
FB (MDF)	Radix-2^4 [11]	$2(\log_4 N-1)$	$2(\log_{16} N-1)$	$2(\log_{16} N)$	$8(\log_4 N)$	$3N/2$	$N/2$	2
FF (MDC)	Proposed, radix-2^2	$2(\log_4 N-1)$	$2(\log_4 N-1)$	0	$4(\log_4 N)$	N	$N/2$	2
FF (MDC)	Proposed, radix-2^3	$2(\log_4 N-1)$	$2(\log_8 N-1)$	$\log_8 N$	$4(\log_4 N)$	N	$N/2$	2
FF (MDC)	Proposed, radix-2^4	$2(\log_4 N-1)$	$2(\log_{16} N-1)$	$2(\log_{16} N)$	$4(\log_4 N)$	N	$N/2$	2
4-PARALLEL ARCHITECTURES								
FF (MDC)	Radix-4, [5]	$3(\log_4 N-1)$	$3(\log_4 N-1)$	0	$8(\log_4 N)$	$8N/3$	$N/3$	4
FF (MDC)	Radix-4, [18], [19]	$3(\log_4 N-1)$	$3(\log_4 N-1)$	0	$8(\log_4 N)$	N	$N/4$	4
FB (MDF)	Radix-2^4, [9]	$4(\log_4 N-1)$	$4(\log_{16} N-1)$	$4(\log_{16} N)$	$16(\log_4 N)$	N	$N/4$	4
FB (MDF)	Radix-2^4, [13]	$4(\log_4 N-1)$	$4(\log_{16} N-1)$	$4(\log_{16} N)$	$16(\log_4 N)$	N	$N/4$	4
FF (MDC)	Proposed, radix-2^2	$3(\log_4 N-1)$	$3(\log_4 N-1)$	0	$8(\log_4 N)$	N	$N/4$	4
FF (MDC)	Proposed, radix-2^3	$4(\log_4 N-1)$	$4(\log_8 N-1)$	$2(\log_8 N)$	$8(\log_4 N)$	N	$N/4$	4
FF (MDC)	Proposed, radix-2^4	$3.5\log_4 N-4$	$4(\log_{16} N-1)$	$3(\log_{16} N)$	$8(\log_4 N)$	N	$N/4$	4
8-PARALLEL ARCHITECTURES								
FF (MDC)	Radix-8, [5]	$6\log_4 N-7$	$7(\log_8 N-1)$	$2(\log_8 N)$	$16(\log_4 N)$	$16N/7$	$2N/7$	8
FF (MDC)	Radix-2, [16]	$8(\log_4 N-1)$	$8(\log_4 N-1)$	0	$16(\log_4 N)$	N	$N/8$	8
FB (MDF)	Radix-2, [7]	$8(\log_4 N-1)$	$8(\log_4 N-1)$	0	$32(\log_4 N)$	N	$N/8$	8
FB (MDF)	Radix-2^4, [8]	$8(\log_4 N-1)$	$8(\log_{16} N-1)$	$8(\log_{16} N)$	$32(\log_4 N)$	N	$N/8$	8
FF (MDC)	Proposed, radix-2^2	$6(\log_4 N-1)$	$6(\log_4 N-1)$	0	$16(\log_4 N)$	N	$N/8$	8
FF (MDC)	Proposed, radix-2^3	$6\log_4 N-7$	$7(\log_8 N-1)$	$2(\log_8 N)$	$16(\log_4 N)$	N	$N/8$	8
FF (MDC)	Proposed, radix-2^4	$7\log_4 N-8$	$8(\log_{16} N-1)$	$6(\log_{16} N)$	$16(\log_4 N)$	N	$N/8$	8

saves 25% of the total number of rotators. Furthermore, the proposed 4-parallel radix-2^4 feedforward FFT saves 50% of the adders and 25% of the W_{16} rotators with respect to radix-2^4 parallel feedback architectures [9], [13].

Finally, the proposed 8-parallel radix-2^k architectures improve on all previous designs in the literature. The proposed 8-parallel radix-2^2 feedforward FFT saves 25% of the rotators with respect to radix-2 feedforward FFTs [16], and 50% of the adders and 25% of the rotators with respect to feedback architectures [7]. The proposed 8-parallel radix-2^3 feedforward FFT reduces the memory requirements and latency of previous radix-8 feedforward FFTs [5], and the proposed 8-parallel radix-2^4 feedforward FFT saves 12% of the W_{16} rotators and 50% of the adders with respect to radix-2^4 parallel feedback designs [8].

VII. EXPERIMENTAL RESULTS

The presented architectures have been programmed for the use in field-programmable gate arrays (FPGAs). The designs are parameterizable in the number of points N, wordlength, and number of samples in parallel P. Table IV shows post-place and route results for different configurations of N and P, using a wordlength of 16 bits. The target FPGA is a Virtex-5 FPGA, XC5VSX240T-2 FF1738. This FPGA includes DSP48E blocks that can be used to carry out mathematical operations. In the proposed designs these blocks have been used to implement complex multipliers that carry out the rotations of the FFT.

Fig. 9 compares the area of the proposed architectures to other equivalent high-throughput pipelined FFTs architectures

TABLE IV
AREA AND PERFORMANCE OF THE PROPOSED P-PARALLEL N-POINT RADIX-2^2 FEEDFORWARD FFT ARCHITECTURES FOR 16 BITS

FFT		Area		Latency	Freq.	Throughput
P	N	Slices	DSP48E	(μs)	(MHz)	(MS/s)
4	16	386	12	0.026	458	1831
	64	695	24	0.081	384	1536
	256	1024	36	0.221	389	1554
	1024	1425	48	1.055	270	1081
	4096	2388	60	6.120	173	693
8	16	688	24	0.025	400	3204
	64	1312	48	0.081	283	2263
	256	1979	72	0.223	242	1937
	1024	2497	96	0.630	249	1995
	4096	3540	120	2.744	200	1598
16	64	2657	96	0.078	245	3921
	256	3754	144	0.151	252	4033
	1024	5044	192	0.406	229	3666
	4096	6423	240	1.516	193	3082

[5], [21] for the same FPGA and synthesis conditions. Full streaming architectures (FS) have been generated using the tool presented in [21], which provides optimized pipelined architectures for a given radix and number of parallel samples. As in previous section, the results in the graphs do not include additional circuits for adapting the input and output data orders.

The results for 4-parallel pipelined architectures are shown in Fig. 9(a). In the figure, the numbers next to the lines indicate the amount of DSP48E slices that each architecture requires. It can be observed that the proposed radix-2^2 architectures require less

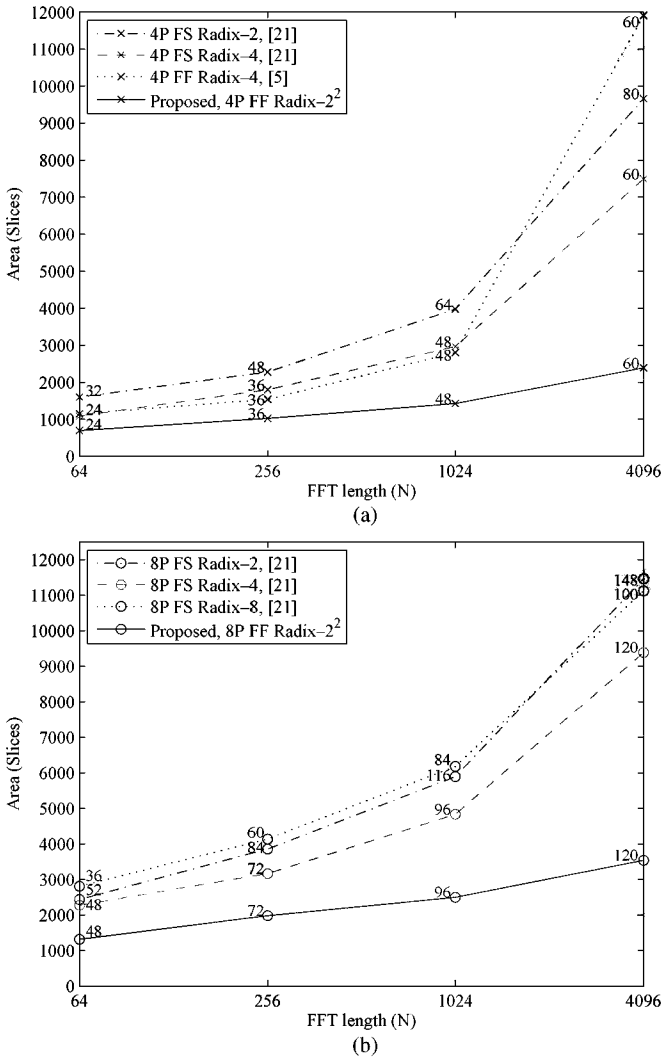

Fig. 9. Area of 4-parallel and 8-parallel pipelined FFT architectures. (a) 4-parallel pipelined FFT architectures. (b) 8-parallel pipelined FFT architectures.

area than previous designs for any FFT size, N. This improvement increases with the size of the FFT. For 8-parallel samples, Fig. 9(b) shows that the proposed designs also improve over radix-2 and radix-4 architectures, and the larger N the larger the savings. Architectures that use radix-8 need less DSP48E blocks at the cost of a significant increase in the number of slices.

Fig. 10 compares the throughput of the proposed designs to other 4-parallel and 8-parallel pipelined FFTs. As can be observed, the proposed designs achieve the highest throughputs both for 4-parallel and 8-parallel designs. Indeed, even higher throughput can be achieved by resorting to 16-parallel radix-2^2 feedforward architectures, as was shown in Table IV.

VIII. Conclusion

This paper extends the use of radix-2^k to feedforward (MDC) FFT architectures. Indeed, it is shown that feedforward structures are more efficient than feedback ones when several samples in parallel must be processed.

In feedforward architectures radix-2^k can be used for any number of parallel samples which is a power of two. Indeed, the

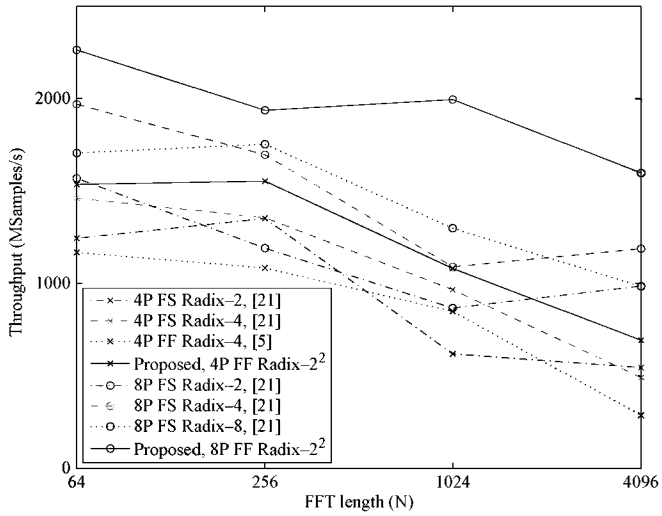

Fig. 10. Throughput of 4-parallel and 8-parallel pipelined FFT architectures.

number of parallel samples can be chosen arbitrarily depending of the throughput that is required. Additionally, both DIF and DIT decompositions can be used.

Finally, experimental results show that the designs are efficient both in area and performance, being possible to obtain throughputs of the order of GSamples/s as well as very low latencies.

Acknowledgment

The authors would like to thank Dr. R. Conway for his valuable suggestions about the presentation of this work.

References

[1] L. Yang, K. Zhang, H. Liu, J. Huang, and S. Huang, "An efficient locally pipelined FFT processor," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 53, no. 7, pp. 585–589, Jul. 2006.

[2] H. L. Groginsky and G. A. Works, "A pipeline fast Fourier transform," *IEEE Trans. Comput.*, vol. C-19, no. 11, pp. 1015–1019, Oct. 1970.

[3] A. M. Despain, "Fourier transform computers using CORDIC iterations," *IEEE Trans. Comput.*, vol. C-23, pp. 993–1001, Oct. 1974.

[4] S. He and M. Torkelson, "Design and implementation of a 1024-point pipeline FFT processor," in *Proc. IEEE Custom Integr. Circuits Conf.*, 1998, pp. 131–134.

[5] M. A. Sánchez, M. Garrido, M. L. López, and J. Grajal, "Implementing FFT-based digital channelized receivers on FPGA platforms," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 44, no. 4, pp. 1567–1585, Oct. 2008.

[6] A. Cortés, I. Vélez, and J. F. Sevillano, "Radix r^k FFTs: Matricial representation and SDC/SDF pipeline implementation," *IEEE Trans. Signal Process.*, vol. 57, no. 7, pp. 2824–2839, Jul. 2009.

[7] E. H. Wold and A. M. Despain, "Pipeline and parallel-pipeline FFT processors for VLSI implementations," *IEEE Trans. Comput.*, vol. C-33, no. 5, pp. 414–426, May 1984.

[8] S.-N. Tang, J.-W. Tsai, and T.-Y. Chang, "A 2.4-GS/s FFT processor for OFDM-based WPAN applications," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 57, no. 6, pp. 451–455, Jun. 2010.

[9] H. Liu and H. Lee, "A high performance four-parallel 128/64-point radix- 2^4 FFT/IFFT processor for MIMO-OFDM systems," in *Proc. IEEE Asia Pacific Conf. Circuits Syst.*, 2008, pp. 834–837.

[10] L. Liu, J. Ren, X. Wang, and F. Ye, "Design of low-power, 1 GS/s throughput FFT processor for MIMO-OFDM UWB communication system," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2007, pp. 2594–2597.

[11] J. Lee, H. Lee, S. I. Cho, and S.-S. Choi, "A high-speed, low-complexity radix- 2^4 FFT processor for MB-OFDM UWB systems," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2006, pp. 210–213.

[12] N. Li and N. P. van der Meijs, "A radix 2^2 based parallel pipeline FFT processor for MB-OFDM UWB system," in *Proc. IEEE Int. SOC Conf.*, 2009, pp. 383–386.

[13] S.-I. Cho, K.-M. Kang, and S.-S. Choi, "Implemention of 128-point fast Fourier transform processor for UWB systems," in *Proc. Int. Wirel. Commun. Mobile Comput. Conf.*, 2008, pp. 210–213.
[14] W. Xudong and L. Yu, "Special-purpose computer for 64-point FFT based on FPGA," in *Proc. Int. Conf. Wirel. Commun. Signal Process.*, 2009, pp. 1–3.
[15] C. Cheng and K. K. Parhi, "High-throughput VLSI architecture for FFT computation," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 10, pp. 863–867, Oct. 2007.
[16] J. A. Johnston, "Parallel pipeline fast Fourier transformer," *IEE Proc. F Commun. Radar Signal Process.*, vol. 130, no. 6, pp. 564–572, Oct. 1983.
[17] B. Gold and T. Bially, "Parallelism in fast Fourier transform hardware," *IEEE Trans. Audio Electroacoust.*, vol. 21, no. 1, pp. 5–16, Feb. 1973.
[18] E. E. Swartzlander, W. K. W. Young, and S. J. Joseph, "A radix 4 delay commutator for fast Fourier transform processor implementation," *IEEE J. Solid-State Circuits*, vol. 19, no. 5, pp. 702–709, Oct. 1984.
[19] J. H. McClellan and R. J. Purdy, "Applications of digital signal processing to radar," in *Applications of Digital Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1978, ch. 5.
[20] M. Garrido, K. K. Parhi, and J. Grajal, "A pipelined FFT architecture for real-valued signals," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 56, no. 12, pp. 2634–2643, Dec. 2009.
[21] P. A. Milder, F. Franchetti, J. C. Hoe, and M. Püschel, "Formal datapath representation and manipulation for implementing DSP transforms," in *Proc. IEEE Design Autom. Conf.*, 2008, pp. 385–390.
[22] Y.-W. Lin and C.-Y. Lee, "Design of an FFT/IFFT processor for MIMO OFDM systems," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 54, no. 4, pp. 807–815, Apr. 2007.
[23] S. Li, H. Xu, W. Fan, Y. Chen, and X. Zeng, "A 128/256-point pipeline FFT/IFFT processor for MIMO OFDM system IEEE 802.16e," in *Proc. IEEE Int. Symp. Circuits Syst.*, 2010, pp. 1488–1491.
[24] M. Garrido, "Efficient hardware architectures for the computation of the FFT and other related signal processing algorithms in real time," Ph.D. dissertation, Dept. Signals, Syst., Radiocommun., Univ. Politécnica Madrid, Madrid, Spain, 2009.
[25] J. W. Cooley and J. W. Tukey, "An algorithm for the machine calculation of complex Fourier series," *Math. Comput.*, vol. 19, pp. 297–301, 1965.
[26] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*. Englewood Cliffs, NJ: Prentice-Hall, 1989.
[27] Y.-N. Chang, "An efficient VLSI architecture for normal I/O order pipeline FFT design," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 55, no. 12, pp. 1234–1238, Dec. 2008.
[28] M. Garrido, J. Grajal, and O. Gustafsson, "Optimum circuits for bit reversal," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 58, no. 10, pp. 657–661, Oct. 2011.
[29] M. Garrido, O. Gustafsson, and J. Grajal, "Accurate rotations based on coefficient scaling," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 58, no. 10, pp. 662–666, Oct. 2011.
[30] F. Qureshi and O. Gustafsson, "Low-complexity constant multiplication based on trigonometric identities with applications to FFTs," *IEICE Trans. Fundamentals*, vol. E94-A, no. 11, pp. 324–326, Nov. 2011.

Mario Garrido (M'10) received the M.S. degree in electrical engineering and the Ph.D. degree from the Technical University of Madrid (UPM), Madrid, Spain, in 2004 and 2009, respectively.

Since 2010 he is a postdoctoral researcher at the Linköping University, Sweden. His research focuses on the design and optimization of VLSI architectures for signal processing applications. This includes the design of hardware architectures for the calculation of transforms, such as the fast Fourier transform (FFT), hardware circuits for data management and the CORDIC algorithm. His research covers high-performance circuits for real-time computation, as well as designs for low area and low power consumption.

J. Grajal was born in Toral de los Guzmanes (León), Spain, in 1967. He received the Ingeniero de Telecomunicacin and the Ph.D. degrees from the Technical University of Madrid, Madrid, Spain, in 1992 and 1998, respectively.

Since 2001, he has been an Associate Professor with the Signals, Systems, and Radio Communications Department, Technical School of Telecommunication Engineering, Technical University of Madrid. His research activities include the area of hardware-design for radar systems, radar signal processing and broadband digital receivers for radar, and spectrum surveillance applications.

M. A. Sánchez Marcos received the M.S. degree in telecommunications with a major in electronics from Universidad Politécnica de Madrid, Madrid, Spain, in 2003, where he is pursuing the Ph.D. degree from the Department of Electronic Engineering.

His research interests include embedded systems and application-specific high-performance programmable architectures.

Oscar Gustafsson (S'98–M'03–SM'10) received the M.Sc., Ph.D., and Docent degrees from Linköping University, Linköping, Sweden, in 1998, 2003, and 2008, respectively.

He is currently an Associate Professor and Head of the Electronics Systems Division, Department of Electrical Engineering, Linköping University. His research interests include design and implementation of DSP algorithms and arithmetic circuits. He has authored and coauthored over 130 papers in international journals and conferences on these topics.

Prof. Gustafsson is a member of the VLSI Systems and Applications and the Digital Signal Processing technical committees of the IEEE Circuits and Systems Society. Currently, he serves as an Associate Editor for the IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS—PART II: EXPRESS BRIEFS and *Integration, the VLSI Journal*. He has served and serves in various positions for conferences such as ISCAS, PATMOS, PrimeAsia, Asilomar, Norchip, ECCTD, and ICECS.